

Application of ILOG Solver in fast intermodal transshipment

Philippe Le Page - FRAMENEC-COGNITECH - Tour Fiat, Cedex 16 - 92084 Paris-La Défense - phone : 33 1 47 96 46 00

Guy Barras - FRAMATOME - Robotics Division - 69000 Lyon - phone : 33 74 72 82 21

1. General Presentation

MARTHA (Mobile Autonomous Robots for Transportation and Handling Applications) is an ESPRIT project which started in 1992. Its ambition to meet the pressing demands of European companies involved in fast intermodal transshipment. Several key techniques and technologies are needed to make the automated evolution of vehicle fleets under some task-level programming or higher level operator control possible, hence being capable of some form of **autonomy** without the current limitation of expensive site preparation. The applications considered are generic and include rail, port and airport intermodal transshipment operations.

Research on mobile robots aims at endowing these machines with a higher level of autonomy enabling them to perform tasks (and in particular their central navigation task) in non-cooperative environments. This objective has yet to be accomplished. However, it is currently possible to obtain a degree of autonomous behaviour by more or less adapting the environment (i.e. making it more cooperative), by simplifying the task itself, or by including a human control at some level. Indeed, in a wide range of applications, it is possible to develop the techniques that are necessary for robot autonomy in a nominal mode of operation.

For many tasks however, the autonomy of the robot cannot be complete: for example, it is beyond the current state of the art to plan and to perform autonomously complex manipulation tasks in a real world situation and in a very limited time. Therefore, several robot architectures provide also for the possibility to include human operator in the control loop, at various levels, ranging from task-level interaction, where some decision is made by a ground-based system, to teleoperation.

In the context of **cargo transport** applications, an additional aspect is the control of a fleet of mobile robots that must cooperate to achieve some global goals in a coherent manner. Several organising schemes have been proposed for similar multi-agent problems. Some are based on a totally free group of agents that communicate their plans. On the other hand, most organisations rely simply on a master-slave relationship, a master sending commands to the other agents. For this project's multi-robot applications, an intermediate hierarchical organisation is considered: a **central control system** is in charge of global mission planning and supervision, and assigns a plan to each robot control system, that executes it with some limited coordination between robots.

2. The Central Control System

The fleet of robots evolves on a site which can be considered to be a network of lanes, connected by crossings or areas. The areas contain stations where the robots can pick up and put down the containers - see figure 1. A station can be a simple free space on the ground where the containers are stored one above the other, or a "stacker" where each container is put inside a slot. In the following, the word "entity" refers to either a lane, a crossing, a station or an area. A lane can be temporarily blocked by an unknown obstacle. A typical mission for a transshipment application consists in the specifications of a global objective, such as "*load station S10 with 10 containers of type T1, picked up in area A1*", or a more specific one such as: "*Tranship containers B1, B2, B3*

and B7 to station S4". Temporal constraints can be added such as: "before 4:30 pm". A robot is supposed to carry only one container at a time. The problem of the stock management of containers is beyond the scope of the project.

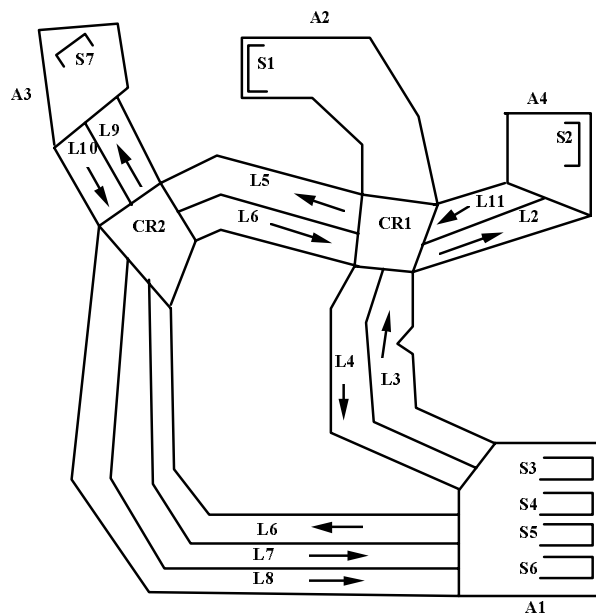


Figure 1: example of a network

In order to ensure a coherent behaviour of the robot fleet, and to prevent critical situations, the overall system's architecture is decomposed into the Central Control System (CCS), and a Robot Control System (RCS) for each robot - see figure 2. The CCS is in charge of planning and supervising the activity of the robot fleet, in order to achieve global objectives given either by an operator or by a higher-level application control system. It is composed of three main sub-systems:

- the Man Machine Interface
- the Multi-Mission Planner (MMP)
- the Global Mission Supervisor (GMS)

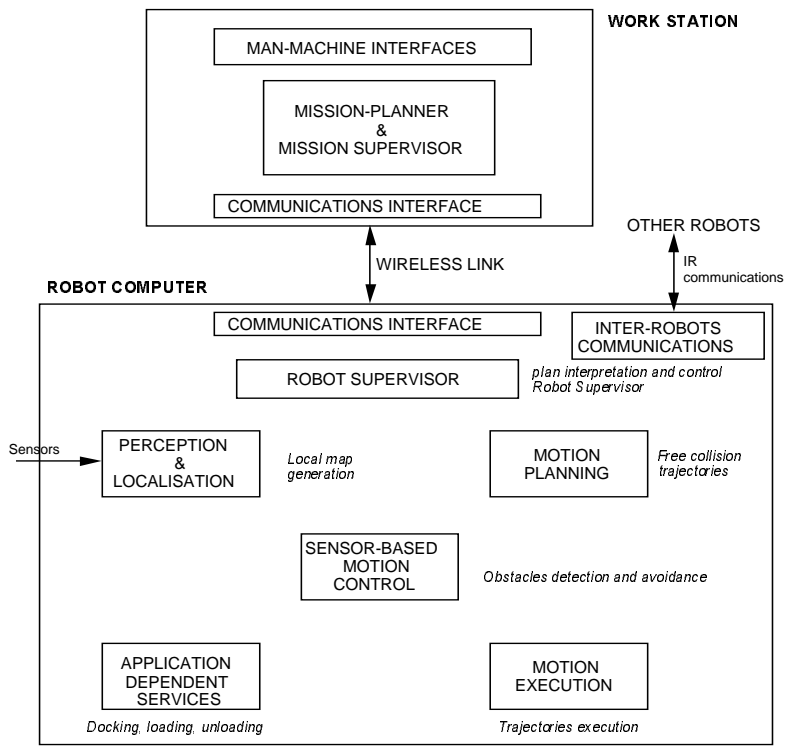


Figure 2: functional architecture

a) The MMP

The role of the MMP is to build a plan for each container which must be transhipped. First, it chooses an available robot to carry the container, then a route for this robot, in order to load and unload it in its loading and unloading station respectively. A route is an ordered list of entities the robot will follow successively. Then the MMP builds a set of Elementary Actions (EA) understandable by each RCS, such as: GOTO C1, GOTO L1, GOTO A2, GOTO S3, DOCK S3, LOAD B1... See figure 3, an example of what a complete plan looks like. The GMS will send each EA to the robot RCS, which will in turn compute the trajectory of the robots inside the entities.

Loading Task	GOTO Area 1	Lane 3	[10 40]
	GOTO Lane 3	Crossing 1	[40 50]
	GOTO Crossing 1	Lane 2	[50 55]
	GOTO Lane 2	Area 4	[55 67]
	GOTO Area 4	Station 2	[67 77]
	DOCK Station 2		[77 79]
	PICK UP Box B3		[79 81]
	UNDOCK Station 2		[81 83]
Unloading Task	GOTO Area 4	Lane 11	[83 93]
	GOTO Lane 11	Crossing 1	[93 113]
	GOTO Crossing 1	Area 2	[113 119]
	GOTO Area 2	Station 1	[119 125]
	DOCK Station 1		[125 127]
	PUTDOWN Box B3		[127 129]
	UNDOCK Station 1		[129 131]

figure 3: example of a plan

When achieving its elementary plans, the MMP has to comply with temporal constraints given in the initial missions coming from the MMI: the plans have to be executed before a certain date. It should also satisfy traffic density constraints in critical regions of the network.

b) The GMS

The mission supervision is in charge of the following functions:

- interactions with the robots, sending robot plans, broadcasting some information about the environment model;
- global plan execution control, updating and monitoring the plan and environment state, the status of each robot and the traffic;
- traffic control and local route modification when necessary;
- exception detection and handling, with the prediction and assessment of event consequences, the management of local resource conflicts, the partial scheduling modification and the replanning request.

Examples of exceptions are robot delay, such as a route that is blocked by an unexpected obstacle, a resource conflict between several robots that they are unable to solve locally, etc. Such exceptions may have consequences on several other robot plans. It is the role of the GMS to predict such consequences, by propagating temporal constraints, in order to maintain the integrity

of the set of plans. However, plans replanning should be avoided when possible, because it is time-consuming. A short-term modification is preferable as a first reaction to an event, and replanning should be performed only when it is necessary (i.e. when the event has long-term effects on several robot plans).

3. Constraint Programming in MARTHA

The objectives sent asynchronously to the CCS require the cooperation of several robots. However, this is only a loose cooperation since each robot acts independently from the others: its elementary plan consists in going to a station, loading a container, going to another station, and unloading the container.

Of course, the robots interfere in the sense that they evolve on the same network: the lanes have limited capacities, a robot may have to wait for another one to cross an area. In fact, we face a resource allocation problem, and we have to deal with temporal and traffic constraints. We have to choose a solution in a quasi-infinite search space and, needless to say, the problem of optimising the overall plans for all the robots is, whatever the optimising criterion, NP-hard. The selected solution was to use constraint propagation techniques. Our project started in 1992, and at that time, ILOG Schedule did not yet exist, so we made a preliminary mock-up with ILOG Solver C++. The first part of our job was to write down all the constraints of our application, secondly to express them using ILOG Solver capabilities and thirdly to delimit the type of solutions we would explore.

a) Constraints in the application

Each container has to be transhipped from its loading station to its unloading station. It can be part of a heap of containers, and have some containers placed above him. Our first type of constraint is of precedence: if $B1$ is above $B2$, $B1$ must be picked up before $B2$.

We have already spoken about temporal constraints: a container has to be picked up (or put down) before a certain date.

All the lanes have limited capacities for the robots, due only to the robot size, the stations can be occupied by only one robot: consequently, we have to deal with "capacity" constraints.

No overtaking is possible in the lanes (they are all one-way lanes): here we have again a sort of "precedence" constraint.

We have already mentioned in this document the traffic density constraints. These are not real constraints, and must be regarded as an optimisation criterion: in order to avoid traffic jams, we must assess the end time of each elementary plan of each robot, and try to make it as small as possible. Doing so, we indirectly guarantee a smooth traffic on the network.

b) Expression of the constraints

The basic element in the MMP and the GMS is the elementary action. Each elementary action has a beginning, an end, and of course a certain duration. It takes place on a network entity (lane, crossing, area or station). We associate two ILOG Solver integer constrained variables to an elementary action, one for its beginning: " st ", one for its end, " et ". The main feature of a constrained variable is its validity domain - here an interval of integers. For us, only the lower bound of the domain will be meaningful: the lower bound of the domain of st will be the time at which the elementary action will begin, the lower bound of the domain of et will be the time at which the elementary action will end. Each elementary action has a pointer *nextPlan* which points to the following action belonging to the plan of the same robot. The et constrained variable of

each elementary action is the *st* of its *nextPlan* elementary action, since the latter begins when the former ends.

A certain number of constraints will be posed and propagated on these two sets of constrained variables. They will depend on the type of network entity the elementary action takes place on. One is common for all types of network entities. It concerns the average time spent by a robot to cross a free network entity. Let *duration* be this time (for a lane, this duration is computed by dividing the length of the lane by the speed of the robot, it is a given estimated constant for the crossings, the areas and the stations), we have then:

$$et > st + duration$$

Of course, if the entity is not free, the time really spent for the crossing will be greater, that is why we don't write:

$$et = st + duration$$

We will now deal with the stations and the lanes. On these entities, each elementary action has a pointer *next* which points to the action which will occur afterwards on the same network entity. So, the list of the elementary actions corresponding to a network entity is ordered in time. In the following, *nextEA* will stand for the action occurring after *EA* on the same entity.

Considering the stations, there are three more constraints to add. A station can contain only one robot at a time:

$$st \text{ of } nextEA > et \text{ of } EA$$

Considering the temporal constraints, let *t* be the date before which the box corresponding to *EA* must be loaded (or unloaded), we have:

$$et \text{ of } EA < t$$

We have also to consider the ordering constraints for the boxes on the stations. For a loading station, the box which is just under the box associated to *EA* will be loaded afterwards. Let *nextEABox* be the elementary action corresponding to the picking up of this box, we have then:

$$st \text{ of } nextEABox > et \text{ of } EA$$

Considering the lanes now, we have to express two types of constraints. No overtaking is possible on a lane:

$$et \text{ of } nextEA > et \text{ of } EA$$

If the lane capacity is *N*, there are at most *N* robots at the same time. Let us call *nextNthEA* the *N*th elementary action following *EA*.

$$st \text{ of } nextNthEA > et \text{ of } EA$$

There are no constraints on capacity for a crossing or an area.

c) Search Algorithm

Each time a set of containers is to be transhipped, the MMP first chooses an available one (e.g. with no containers above), and selects the robots which will be available before a certain date $T+\Delta T$, which means that they have no plans or if this is not the case, that their last plan will end before $T+\Delta T$.

Then, it builds and deletes several plans for each robot, following several routes in the network, joining first the loading station, then the unloading station of the container. The plans are made

incrementally, each EA is created, its constraints are activated, and its ending time "*et*" automatically computed. The plans which has the earliest ending time is then rebuilt.

It follows from this description that the MMP does not look for the optimal set of plans, since it does not try the complete set of solutions. However, due to the uncertainties concerning the robots' speed (which is of course not constant, contrary to our assumption used for the computation of the EA duration), the temporal uncertainties concerning the docking and loading operations, and taking into account that these plans can be invalidated at any time (because of a robot failure, the detection of an obstacle on a lane...), the MMP need not look for optimal plans.

For the same reasons, the MMP does not make a plan for the overall set of containers which have to be transhipped, but makes sure that all the robots have to execute a plan ending after a certain time interval. The GMS and the MMP run concurrently. Both processes communicate via a database where every information concerning the network, the containers, the robots and their plans, the missions, are stored. As soon as a robot is about to finish the execution of its last plan, if there are still containers to tranship, the GMS triggers the MMP, waits for the end of the planning (unless an urgent message is sent by one of the robot, or a new mission is sent by the MMI, in which case, the GMS interrupts the MMP), and then reads in the database the new plans.

The GMS updates continuously the plans, taking into account the messages from the robot. Each time a robot achieves an EA, it sends a report to the GMS, which deletes the EA and sets the current time as the new beginning date for the next EA of the same plan. Then the ILOG Solver propagation mechanism automatically updates all the beginning and ending times of the EA, for all the robots. If a temporal constraint is violated (a robot is late, or will put down its container too late...), the GMS deletes only the plans which need to be recomputed. If a robot fails in performing an EA, or warns the GMS that a lane is blocked, the process is the same. The MMP is triggered afterwards, and makes new plans.

4. Conclusion

The Martha project is nearing its end and the CCS is already successful in handling simulated robots. Exceptional events are treated in real time, the GMS recovers in an efficient way and the estimations of the duration of each plan are updated continuously, giving the operator a good view of the evolution of the situation.

Thus, Ilog Solver C++ has proved to be a robust tool, able to deal with a great variety of constraints, and to solve real time planning problems. The authors were among the first users of Ilog Solver C++ and would like to thank the Ilog Solver team for their precious advice during the conception of the CCS and their support during the project's lifetime.